BÁO CÁO BÀI TẬP 2

Môn học: Phương pháp học máy trong an toàn thông tin

Tên chủ đề: Reinforcement Learning

GVHD: Phan Thế Duy

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: .....

STT	Họ và tên	MSSV	Email
1	Trần Hoàng Khang	19521671	19521671@gm.uit.edu.vn
2	Nguyễn Tú Ngọc	20521665	20521665@gm.uit.edu.vn
3	Lê Hồng Bằng	19520396	19520396@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Phân tích chung về thông tin môi trường	80%
1.1	Xuất ra các thông tin yêu cầu về môi trường (số lượng không gian actions, states có thể, …) + Giải thích	100%
1.2	Lưu các thông tin trên vào file CSV	0%
2	Sử dụng mô hình huấn luyện để tìm đường đi tốt nhất và báo cáo lại kết quả	100%
2.1	Mô hình hóa bằng phương pháp **MDP (Markov Decision Process)**	100%
2.2	Sử dụng thuật toán **Q-Learning**	100%
2.3	Lưu lại video giải dựa trên hành động của agent	100%
3	Đánh giá được hiệu suất của phương pháp học tăng cường mang lại.	100%
3.1	Xây dựng lại model dưới dạng hàm để dễ tái sử dụng (cái này tự thêm :>)	100%
3.2	Chạy 1000 lần và xuất kết quả đánh giá	100%

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

Note: Mình break 2 task trong bài tập nhỏ ra thành 3 phần như dưới đây cho rõ ràng. Giải thích mình đã lưu rất kỹ trong file Notebook nên ở đây mình không nhắc lại mà chỉ mang tính chất báo cáo.

Note: Task lần này của chúng ta là sử dụng thuật toán và mô hình cụ thể để áp dụng lên trò chơi Taxi (version 3) có sẵn trong thư viện OpenAI Gym. Tận dụng mẫu tham khảo đã cho trước với môi trường 'LunarLander-v2' sẽ rất thuận tiện.

1. Phân tích chung về thông tin môi trường

```
The observation space: Discrete(500)
The action space: Discrete(6)
```

Số lượng các trường hợp không gian mẫu rời rạc của observation là **500**

Số lượng trường hợp không gian mẫu rời rạc của action là **6**

**Phần lưu thông tin vào file CSV chưa làm, phần này mình không hiểu để lưu như thế nào :')*

2. Sử dụng mô hình huấn luyện để tìm đường đi tốt nhất và báo cáo lại kết quả

- Sử dụng mô hình MDP để hiện thực training theo hình vẽ dưới:

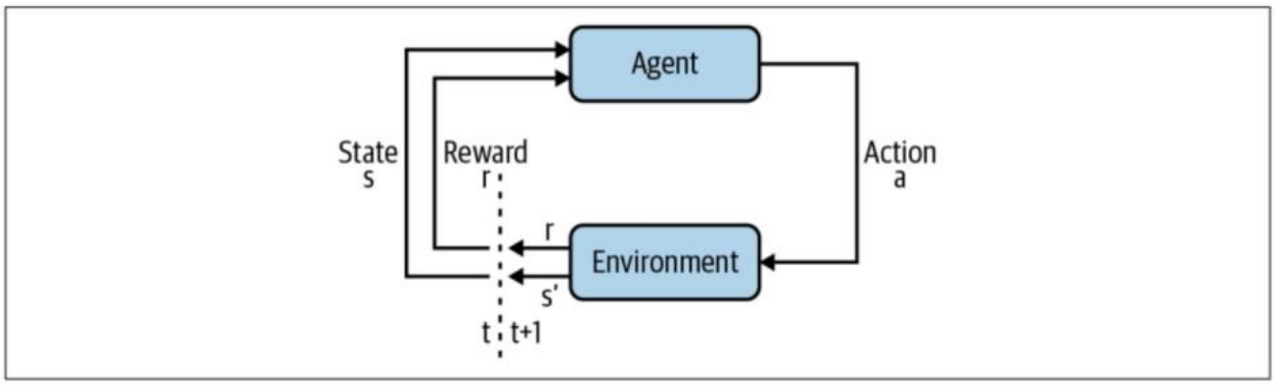


Figure 2-4. A representation of the interface and feedback loop between the environment and the agent in an MDP. The agent accepts the current state and reward from the environment and based upon this information performs an action. The environment takes the action and produces a new reward and state for the next time step.

Với mỗi action agent thực hiện, tương tác (interact) lên môi trường thì đồng thời môi trường sẽ cho ra một state (trạng thái) của nó và reward (phần thưởng) tương ứng với hành động đó và `gửi lại thông tin này cho agent để nó nhận biết và phục vụ cho việc training.

- Sử dụng thuật toán **Q-Learning**:

+ Sử dụng bảng Q-table để cập nhật giá trị tối ưu Q-values (ý nghĩa của bảng được giải thích trong file *Notebook*). Đầu tiên ta khởi tạo bảng với các phần tử bằng 0.

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

+ Sau đó, thông qua việc **exploration** và **exploitation**, agent sẽ cập nhật được bảng với điều kiện dừng là khi trò chơi taxi hoàn thành xong: xe đã đón khách đúng nơi và đưa khách đến đúng chỗ

```
for s in range(max_steps):

    # exploration-exploitation tradeoff
    if random.uniform(0,1) < epsilon:
        # Exploration
        action = env.action_space.sample()
    else:
        # Exploitation
        action = np.argmax(qtable[state,:])

    # take action and observe reward
    #new_state, reward, terminated, truncated, info = env.step(action)
    new_state, reward, done, info = env.step(action)
    print("New state: %d" % new_state)
    print("Reward: %d" % reward)
    print("Done: %r" % done)
    print("Info: %s" % info)

    # Q-learning algorithm
    qtable[state,action] = qtable[state,action] + learning_rate * (reward + discount_rate * np.max(qtable[new_state,:])-qtable[state,action])

    # Update to our new state
    state = new_state

    # if done, finish episode
    #if terminated == True or truncated == True:
    if done == True:
        break
```

+ Chúng ta sẽ cho agent học với **1000 episodes** và liên tục update bảng:

Note: Trong file tham khảo cho agent thực hiện với 10000 episodes nhưng mình thấy không cần thiết và mất rất nhiều thời gian nên mình giảm xuống 1000 cho nhẹ ^_^, kết quả về sau vẫn khá tốt

```
New state: 259
Reward: -1
Done: False
Info: {'prob': 1.0}
New state: 279
Reward: -1
Done: False
Info: {'prob': 1.0}
New state: 379
Reward: -1
Done: False
Info: {'prob': 1.0}
New state: 479
Reward: -1
Done: False
Info: {'prob': 1.0}
New state: 475
Reward: 20
Done: True
Info: {'prob': 1.0}
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
TRAINING PROCESS COMPLETED OVER 1000 EPISODES
```

Q-table của chúng ta sẽ có dạng như sau:

```
[74] q_table

array([[  0.        ,   0.        ,   0.        ,   0.        ,
          0.        ,   0.        ],
       [ -4.12190717,  -3.97792957,  -3.01007477,  -3.97792957,
         -1.6445568 , -11.0840809 ],
       [ -3.40448016,  -3.14630078,  -3.47506913,  -3.14630078,
          3.192     ,  -9.        ],
       ...,
       [ -2.7393912 ,  -2.7447948 ,  -2.7393912 ,  -3.26572394,
         -9.        ,  -9.        ],
       [ -3.75357265,  -3.83585988,  -3.75357265,  -3.72722459,
         -9.        ,  -9.        ],
       [ -1.638     ,  -1.70928   ,  -1.638     ,  11.97      ,
         -9.        ,  -9.        ]])
```

Công thức tính Q-values ở trên là mình hoàn toàn follow theo, đây là công thức đặc trưng của thuật toán.

- Chạy thử game với agent đã được train và quan sát cách agent xử lý:

```
TRAINED AGENT
Step 1
+---------+
|R: | : :G|
| : | : : |
| : :[ ]: : |
| | : | : |
|Y| : |B: |
+---------+
  (South)

score: -1
TRAINED AGENT
Step 2
+---------+
|R: | : :G|
| : | : : |
| :[ ]: : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)

score: -2
```

```
TRAINED AGENT
Step 3
+---------+
|R: | : :G|
| : | : : |
|[ ]: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (West)

score: -3
TRAINED AGENT
Step 4
+---------+
|R: | : :G|
| : | : : |
| : : : : |
|[ ]| : | : |
|Y| : |B: |
+---------+
  (South)

score: -4
```

```
TRAINED AGENT
Step 5
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)

score: -5
TRAINED AGENT
Step 6
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)

score: -6
```

```
TRAINED AGENT
Step 7
+---------+
|R: | : :G|
| : | : : |
| : : : : |
|_| : | : |
|Y| : |B: |
+---------+
  (North)

score: -7
TRAINED AGENT
Step 8
+---------+
|R: | : :G|
| : | : : |
|_: : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)

score: -8
```

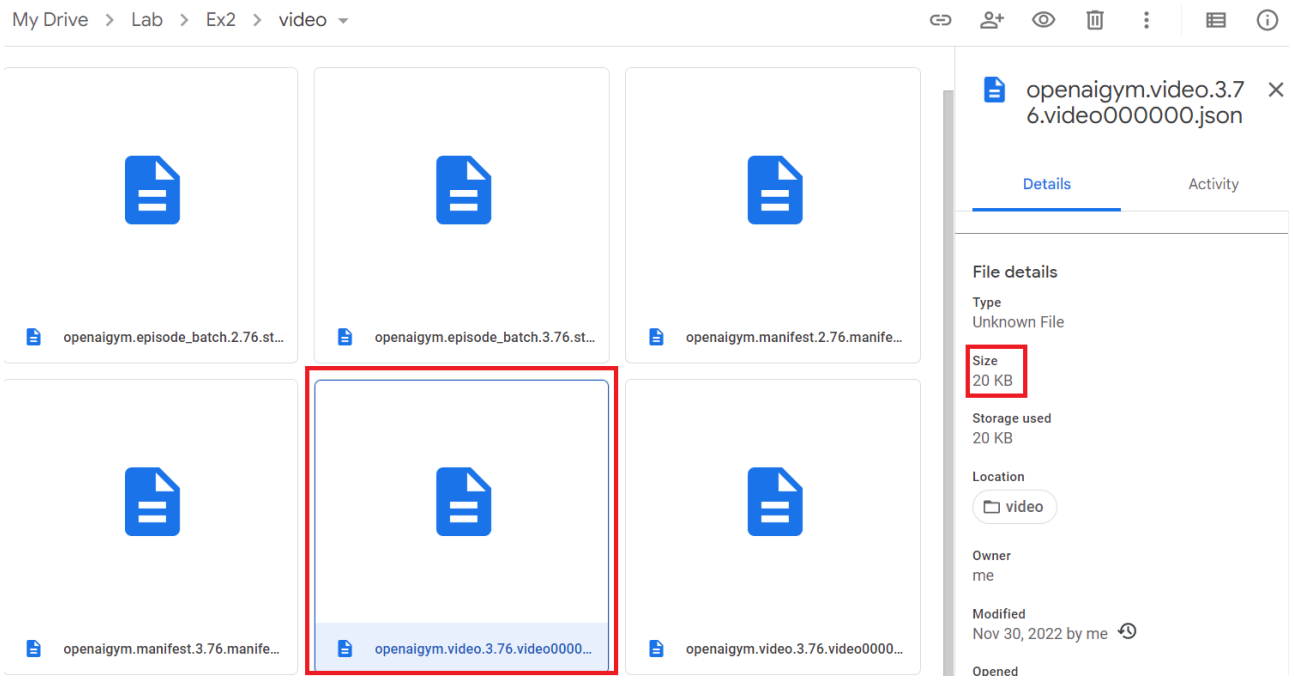
```
TRAINED AGENT
Step 9
+---------+
|R: | : :G|
|_: | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)

score: -9
TRAINED AGENT
Step 10
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)

score: -10
TRAINED AGENT
Step 11
+---------+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)

score: 10
```

**Phần lưu quá trình này vào thành một video mình làm nhìn có vẻ không đúng lắm. Nhưng chúng ta vẫn có thể visualize và hình dung theo cách này <3*

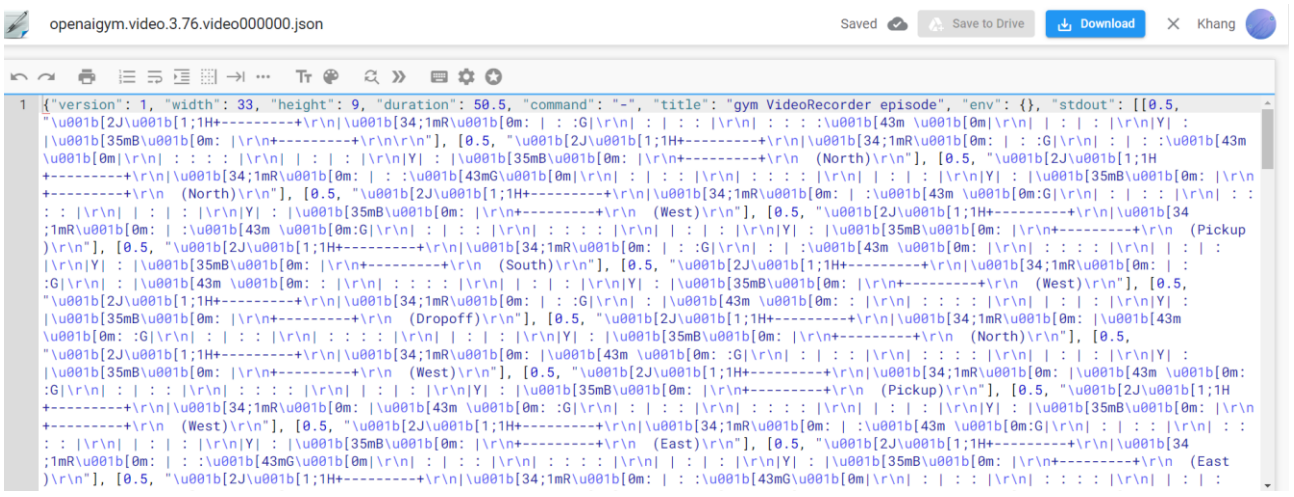
Dùng Wrappers **Monitor** (hoặc *RecordVideo*, *monitoring*) để lưu lại dưới dạng video. Code sample như sau:

```python
from gym.wrappers import Monitor

save_video_path = '/content/drive/MyDrive/Lab/Ex2/video'

# create Taxi environment and prepare the path to store output video
env = Monitor(gym.make('Taxi-v3'), save_video_path, force=True)
```

Sau khi chạy xong lại code xem cách chơi của một agent đã train thì sẽ có video lưu lại quá trình này cùng với các file metadata cho từng episode



Sau đó ta xem thử file "video" (mình thấy nó chả giống video chút nào :v) là file có dung lượng lớn nhất. Nhưng video của chúng ta chỉ lưu dưới dạng dữ liệu **ANSI**



Do vậy nên mình thấy chạy render file ANSI này trên terminal của máy local của mình thì map state chuyển cảnh liên tục thì nhìn nó mới có motion, còn trên colab thì hiện tại mình chưa biết xử lý sao, nó cứ in lần lượt ra từng map state thôi.

-> Nhưng mình cứ công nhận này là video đi, vì mình search cháy máy ra mỗi cách này 😊

3. Đánh giá hiệu suất của phương pháp học tăng cường mang lại.

Phần này mình hoàn toàn follow step-by-step với file tham khảo đã cho trước, tạo các hàm để tái sử dụng và chạy nhiều lần:

- Tạo hàm sử dụng thuật toán **Q-Learning**

```
[72] def q_learning(env, num_episodes, max_steps, learning_rate, discount_rate, epsilon, decay_rate):
        q_table = np.zeros((env.observation_space.n, env.action_space.n))
        rewards_all = []
        for episode in range(num_episodes):
            state = env.reset()

            reward_episode = 0.0
            done = False
            # epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp(-decay_rate*episode)
            for step in range(max_steps):
                exploration = random.uniform(0,1)
                if exploration < epsilon:
                    action = env.action_space.sample()
                else:
                    action = np.argmax(q_table[state, :])

                next_state, reward, done, info = env.step(action)
                q_table[state, action] = q_table[state, action] * (1 - learning_rate) + learning_rate * (reward + discount_rate * np.max(q_table[next_state,:]))

                reward_episode += reward
                state = next_state

                if done:
                    break
            rewards_all.append(reward_episode)
        print(f'Episode {episode} finished')
        return q_table, rewards_all
```

- Tạo hàm chạy game với agent đã train:

```
def play(env, q_table, render=False):
    state = env.reset()
    total_reward = 0
    steps = 0
    done = False
    while not done:
        action = np.argmax(q_table[state, :])
        next_state, reward, done, info = env.step(action)
        total_reward += reward
        steps += 1
        if render:
            env.render()
            time.sleep(0.2)
            if not done:
                display.clear_output(wait=True)
        state = next_state

    return (total_reward, steps)
```

- Tạo hàm chạy nhiều lần:

```
def play_multiple_times(env, q_table, max_episodes):
    success = 0
    list_of_steps = []
    for i in range(max_episodes):
        total_reward, steps = play(env, q_table)

        if total_reward > 0:
            success += 1
            list_of_steps.append(steps)

    print(f'Number of successes: {success}/{max_episodes}')
    print(f'Average number of steps: {np.mean(list_of_steps)}')
```

Kết quả về độ hiệu quả train của agent:

```
play_multiple_times(env, q_table, 1000)
```

```
Number of successes: 992/1000
Average number of steps: 13.223790322580646
```